Semantic Turkey: A Browser-Integrated Environment for Knowledge Acquisition and Management

Maria Teresa Pazienza, Noemi Scarpato, Armando Stellato and Andrea Turbati
University of Tor Vergata, Dept. of Computer Science, Systems and Production, Via del Politecnico 1, Rome, 00133, Italy

Abstract. Born four years ago as a Semantic Web extension for the Web Browser Firefox, Semantic Turkey pushed forward the traditional concept of links&folders-based bookmarking to a new dimension, allowing users to keep track of relevant information from visited websites and to organize the collected content according to standard or personally defined ontologies. Today, the tool has broken the boundaries of its original intents and can be considered, under every aspect, an extensible platform for Knowledge Management and Acquisition. The Semantic Bookmaking and Annotation facilities of Semantic Turkey are now supporting just a part of a whole methodology where different figures, from domain experts to knowledge engineers, can cooperate in developing, building and populating ontologies while navigating the Web.

Keywords: Semantic Browsing, Semantic Annotation, Semantic Bookmarking, Ontology Development

1. Introduction

The Semantic Web is becoming ever and ever a concrete reality: with SPARQL reaching W3C recommendation in 2008 [1], languages for data representation and querying have finally completed standardization, closing the gap on usability of Semantic Web technologies in real case scenarios. At the same time, initiatives such as Linked Open Data [2] have boosted the process of data provisioning on the Web. Finally, interests and research in SW technologies have definitely migrated from mere ontology development (which has now met industry standards) to the discovery and provision of applications which can exploit full Semantic Web potential: homogenous access to distributed information providers, connecting conceptual and information resources on the Web, Open Data.

With this scenario in mind, we have worked towards the definition of a Semantic Web browser extension which is two-fold in its offer: first, it is of interest for ontology developers and domain experts (since it aims at facilitating the process of knowledge acquisition and development even for non-technology-savvy users), second, it provides an extensible infrastructure over which SW applications, needing and relying on rock-solid web browsing functionalities as well as on RDF management capacities, can be developed and deployed.

These objectives have been pursued during a two-years work of finalization and reengineering of Semantic Turkey [3], a Semantic Web extension for the popular Web Browser Firefox [4].

In this issue, we describe the original application for Knowledge Acquisition and Management, and introduce and discuss the main innovations which transformed the new incarnation of Semantic Turkey into an open and extensible platform for Semantic Web Development.

2. Related works

Due to the multifaceted nature of our platform, an overview of related research should embrace diverse fields such as Ontology Editing and Visualization,
Semantic Web Browsing, (Social/Semantic) Bookmarking solutions and Semantic Annotation. In this section we recall the main works in these areas, and provide insight readings for a thorough view.

Regarding Ontology Editing, probably the most used and widely known Ontology Editing Platform is Protégé [5,6]. Realized at the Center for Biomedical Informatics Research of the University of Stanford, Protégé has been for years the leading environment for Ontology Management and has also contributed to the first spread of Semantic Web Technologies in research communities and industries. The Protégé project is currently active, with the Stanford team carrying on development and maintenance of Protégé 3.x, and the University of Manchester developing the next version: Protégé 4.x, which is still in beta development. Another interesting framework is offered by the Neon toolkit [7]: an extensible Ontology Engineering Environment, which has been developed inside the homonymous IP project co-funded by the European Commission’s Sixth Framework Programme. Today, Ontology Development has reached industry standard, as witnessed by commercial off-the-shelf products such as [8].

Regarding Information Visualization through Semantic Web technologies, or “Semantic Browsing”, the first reference which comes to mind is probably the Haystack web client [9]. Developed at the MIT laboratories, was conceived as an application that could be used to browse arbitrary Semantic Web information in much the same fashion as a Web browser can be used to navigate the Web. In this sense, Haystack is more an RDF-based extensible Information Management System for various domains (collecting pictures, organizing contacts…) then a general Knowledge Management and Acquisition system: standard point-and-click semantics let the user navigate over aggregation of data projected from RDF repositories sparse over different arbitrary locations. The application has been built as an extension for the popular Integrated Development Environment Eclipse [10]; this choice facilitates extension of the tool thanks to Eclipse flexible plug-in mechanism, but requires the user to adopt Eclipse as a platform for browsing the web and collecting data from it: a negative aspect for the average user, who would just prefer to rely on his trusted personal web browser and try out other features which are not too invasive for his usual way of working.

An opposite approach is being followed by Magpie [11], which is deployed as a plug-in for the Microsoft Internet Explorer Web Browser. In its first incarnation, Magpie allowed for semantic browsing, intended as the parallel navigation of purely “exposed” web content and of its associated semantic layer (an ontology associated to the web resource, which semantically describes its content). Magpie also allows for collaborative semantic web browsing, in that different persons may gather information from the same web resource and exchange it on the basis of a common ontology. Later work on Magpie [12] extended the platform more and more towards the vision of the Semantic Web as “an open web of interoperable applications” [13], by allowing bi-directional exchange of information among users and services, which can be opportunistically located and composed, either manually (web services) or automatically (semantic web services).

From (part of) the same authors of Haystack, comes Piggy-Bank [14], an extension for the Firefox web browser [4] that lets Web users extract individual information items from within web pages and save them in RDF, replete with metadata. Piggy Bank then lets users make use of these items right inside the same web browser. These items, collected from different sites, can then be browsed, searched, sorted, and organized, regardless of their origins and types. Piggy-Bank users may also rely on Semantic Bank, a web server application that lets them share the Semantic Web information they have collected, enabling, as for Magpie, collaborative efforts to build sophisticated Semantic Web information repositories from daily navigation through their enhanced web browser.

Thinking about the “bookmarking” aspects of Semantic Turkey, we mention here trends in “social bookmarking” tools and services. The most popular one, del.icio.us (http://delicious.com/), is a service for building personal collections of bookmarks and access them online. It is possible, through the same service, to add links to a collection of bookmarks, to categorize the related sites with keywords, and to share the personal collection with other users. Google offered a similar solution with its Google Notebook [15]. The idea is quite simple: open a scratch electronic paper from within your web browser, and let the user add not only bookmarks, but write complete multimedia comments (by using Google Page Creator technology [16]), which can also be shared with other people (currently, this sharing service is not yet active).

Lastly, we mention here a few outcomes in the field of Semantic Annotation. Research on this field is mainly addressing three aspects: how to set up an annotation environment, how to improve the process and extend to several media, and how to automate it.
The Annotea W3C project [17], suggests RDF based standards for representation of annotations, and provides a general architecture for establishing client-server annotation frameworks. Several clients have been developed for this architecture, such as Amaya [18] and Annozilla [19]. Melita [20] and KIM [21] are probably the most prominent examples of applying decades of research on NLP to automate the Semantic Annotation. AKTive Media [22], the successor of Melita, pushes forward the concept of annotation to cover different media other than text. A thorough overview on Semantic Annotation can be found in [23].

What lacks from the previous approaches is a really integrated solution which is able to combine the best of worlds from visualization, annotation and ontology development, possibly by extending the web browser, which still represents the best (and robust) solution for accessing web content.

Also, as remarked in [23], about annotation tools, though “there are signs that annotation systems are giving users more control of ontologies”, “ontology maintenance [...] is poorly supported, or not supported at all, by the current generation of [semantic annotation] tools”.

Seen from the other side (ontology development tools), the RDF family (RDF, RDFS, OWL, SKOS) of models as well as many standard vocabularies such as Dublin Core, offer properties providing meta-knowledge about what is behind the creation of resources in an ontology (such as the RDF rdfs:seeAlso, or Dublin Core dc:relation, dc:source and dc:subject). This is because the specification of a domain should be naturally connected with the process of acquiring knowledge from external sources, and thus of documenting references to them, to better qualify the nature of formalized concepts. However, ontology development tools seem to live in a world of mere algebraic representation, requiring lot of hand work or parallel use of different tools if different figures need to cooperate and make reference to existing information (re)sources.

Semantic Turkey differentiates from similar, previously described approaches, by mixing ontology development functionalities with the ease of use of a system for acquiring knowledge from the web. This way, instead of working on different frameworks and producing different kind of data which need to be integrated, domain experts may start to sketch ontologies and keep track of the information they get from the web, leave comments and references which can be reused and examined by knowledge engineers in continuous refinement circles.

3. Motivations

Semantic Turkey had been initially developed as a prototype for a Web Browser extension with advanced bookmarking capabilities [24]: its mission was to go beyond the purely partitive semantics of traditional links&folders bookmarking, and promote a new paradigm, aiming at “a clear separation between (acquired) knowledge data (the WHAT) and their associated information sources on the web (the WHERE)”.

We thus coined the expression Semantic Bookmarking to indicate the process of eliciting information from (web) documents, to acquire new knowledge and represent it through knowledge representation standards, while keeping reference to the original information sources.

The main difference with Semantic Annotation resides in the focus: the term “Semantic Annotation”, though being subject (as underlined in [21]) to slightly different interpretations, which are in some cases too much bound to the specific research settings where the term has been adopted (e.g. in [25,26] and, again, in [21]), has converged in literature towards the definition of “the process of associating portions of text of analyzed documents to predefined sets of semantic descriptors”. So, the text is the focus of Semantic Annotation, whereas the first objective of Semantic Turkey was (and still is) to facilitate users in acquiring and organizing their knowledge, while keeping at the same time references to the source of information which are being consulted. Also, in a ever-changing setting as the WWW, keeping and maintaining precise reference (pointers to position in documents) to textual content would produce information doomed to corrupt, due to modifications of the bookmarked pages: for this reason, pointers to pages as a whole (i.e. bookmarks) were considered the good compromise for this kind of task.

This idea thus translated into a series of functionalities for the user which, through very easy-to-use drag’n’drop gestures, could select textual information from web pages, create objects in a given domain and annotate their presence in the web by keeping track of the selected text and of its provenience (web page url, title

2Though traditional Semantic Annotation is still made possible thanks to extensions thought for this, such as: http://semantic turkey.uniroma2.it/extensions/rangeannotator/
etc…). An example is given in Fig. 1 where the user is adding the musician Steve Morse as an object in his ontology, while at the same time decorates it with a bookmark to his homepage and provides further details about him (the instrument he plays, the musical genre etc…) getting them from that same page.

4. From Semantic Bookmarking to Knowledge Management and Acquisition

Standing on top of mature results from research on Semantic Web technologies, such as Sesame [27] and OWLim [28] as well as on a robust platform such as the Firefox web browser, Semantic Turkey differentiated from other existing approaches which are more specifically tailored respectively towards knowledge management and editing [5], semantic mashup and browsing [12,14] and pure semantic annotation [20,17] by introducing a new dimension which is unique to the process of building new knowledge while exploring the web to acquire it.

By focusing on this aspect, we went beyond the original concept of Semantic Bookmarking and tried to amplify the potential of a complete Knowledge Management and Acquisition System: we thus aimed at reducing the impedance mismatch between domain experts and knowledge investigators on the one side, and knowledge engineers on the other, providing them with a unifying platform for acquiring, building up, reorganizing and refining knowledge.

5. User Interaction

The final project moved to an open editor for data modeled upon languages of the RDF family, allowing the exploitation of almost all of those language potentialities (currently, it does not allow editing of complex OWL descriptions, though it loads them and reasoners exploit their content; also, SKOS and SKOS-XL editing will be provided with the next release). To allow maximum flexibility, every element in the ontology can now be added through the advanced bookmarking/annotation functionalities (see Fig. 2) or directly through the ontology editor (in both cases, further annotations can be added later to the created objects).
Fig. 2 Activity diagram for semantic bookmarking/annotation
Fig. 2 shows the different annotation/knowledge acquisition possibilities offered by the functionalities based on integration with the hosting web browser: the process is multifaceted in its possible outcomes, though very easy to carry on, since it depends on implicit, contextual factors, such as where in the ontology the user drops the element dragged from the page, as well as on simple interaction steps with the user (like choosing if adding new annotations for a previous element or adding a value for a property, followed by further possibilities depending on the kind of property).

5.1. “Macroing” series of ontology editing operations

The drag’n’drop features for capturing data have been conceived to speed up the process of knowledge acquisition, allowing for complex series of ontology editing operations to be implicitly executed, depending on the specific action performed by the user. In the previous example, if we drag “Deep Purple” over the musician Steve Morse, and then select the playsInBand object property, the following update operations on the underlying ontology are being performed:

- the creation of a new instance (the object of the relation) with local name “DeepPurple” (taken after the selected text), if this is specified as a new resource not already existing in the ontology
- the assertion of a relation (identified by the chosen object property) between the selected object (the “Deep Purple” band) and the instance where the text has been dropped (Steve Morse)
- the assertion of the instance-of relation between the object of the above relation and the class selected from the range of the object property (e.g. DeepPurple as a MusicBand, or even RockBand, because the user is prompted with class-trees rooted on classes featured in the ranges of the selected object property)
- the creation of the bookmarked web page (as an individual in the ontology) and its associated data (title, url etc...)
- the creation of a semantic annotation linking the newly created individual with the bookmarked web page

at the cost of just a drag&drop and a couple of intuitive choices among those proposed through the acquisition process.

Finally, actions taken by the tool following user’s gestures are described as extensible objects, which can be changed, modified, replaced etc. (see “client extendibility” paragraph on section 7.2).

5.2. Real “Open World Assumption”-Aware Approach to User Interface

Whereas a constraint-checking approach to user interfaces would exploit constraints defined in the underlying data model as a strict base for populating form-filling panels, not allowing any operation which could invalidate the constraints, a tool whose knowledge model is based on the open world assumption and on inferential capabilities, uses constraints to just suggest values to the user, or to optionally remove palely incompatible values (that is, values which, by inference, would produce an inconsistency in the model) from choice lists, and let in any case complete freedom to users. For example, when, by following a drag&drop action, a value needs to be added to a resource, the range of suggested properties is first selected on those whose rdfs:domain is computed by inference to include at least one of (and be compatible with all of) the types of the subject resource. Much the same way, when a property has been selected for adding a value, resources can be selected from a class tree-view rooted on the rdfs:range of the property (with analogous considerations to the previous case). These suggestions can be bypassed (e.g. asking to display all the properties, or to explore the whole class tree instead of the suggested part), in that the user can go out of available boundaries, and introduce new “implicit” knowledge by adding ground facts which alter, by inference, the knowledge of the domain. This kind of interaction surpasses the limitations of (at least some of) current ontology editing tools, which are still not fully acquainted with the inferential aspects of the OWL language. For example, Protégé OWL 3.x [6], though offering advanced features and wizards for assisting users in adding entries to an ontology, is still bound to its original constraints-based model [5] which binds subject and object values of triples to the defined rdfs:domain and rdfs:range of the predicate. Protégé 4\footnote{http://protege.stanford.edu/download/registered.html#p4}, being completely targeted for the OWL standard, abandons this constrained approach, though editing of proper-
ty-value editing is still in its infancy and, at present time, its authors preferred to not address at all classification-related issues and to show instead the (whole) list of available instances when the user asks for potential values to be added to object properties. Semantic Turkey thus makes ontology editing faster by proposing suggestions to the user, which rely on declared restrictions and on asserted (or inferred) types and values, but he can always break these boundaries and have access to the whole data, eventually letting further inference follow its actions.

5.3. Other features

To allow for maximum flexibility, the kind of interaction with the user is not limited to semantic bookmarking. Users can thus browse and edit (Fig. 3) the ontology (both knowledge model and instance data) by using Semantic Turkey like any other ontology editing tool.

Native UI: All ontology editing operations are carried out through buttons and context menus associated to the nodes of the tree view, in a way much similar to other ontology editing tools. Unlike other approaches to ontology management capabilities embedded in web browser (such as Piggy-Bank [14]), which rely on Web Based rendering of user interfaces, Semantic Turkey offers complete interaction with the ontology via the XUL interface completely integrated in the browser. The user is thus not diverted from web navigation (i.e. the main browser panel is still focused on the visited web page, which would otherwise be replaced by the HTML UI) and may, at the same time, maintain focus over both the observed web page and the ontology.

Semantic Navigation. As an additional feature, the user may graphically explore the ontology, thanks to the SemanticNavigation component: a customized version of the TouchGraph library [29]. A Java applet
will be loaded on a new tab of the browser, displaying the graph view of the ontology, allowing the user to navigate its content. The nodes of the graph will be displayed in different manners, according to the nature of the ontological entity: classes, properties or individuals. By dragging the mouse pointer on a node that represents an individual, it is possible to popup a window, which contains the URLs of the pages where that instance has been annotated.

6. Knowledge Model

With respect to its former prototype, Semantic Turkey offers now a complete ontology editing environment, allowing users to import, edit and merge ontological data coming from different RDF/OWL sources. The former three tiered KM (see [24]) has thus been abandoned in favor of a single separation between the explicit (domain) knowledge managed by the user and the one which guides system’s behavior. This last layer, defined as the Application Ontologies Layer, is kept invisible to the user, and is only exploited by the application to drive it knowledge based functionalities. Semantic Turkey currently includes one vocabulary in this layer, the Annotation Ontology: a set of concepts (and related properties) which are used to keep track of user annotations from the web. These include:

- ann:WebPage (rdfs:subClassOf ann:Document) concept for storing information about the annotated pages (such as ann:URL and ann:title), that is, the pages where part of the text is annotated with respect to the ontology and thus added to it as a new individual

- ann:SemanticAnnotation containing the annotations performed by the user, and described by the bookmarked ann:WebPage, resource etc… these can be both ann:TextualAnnotation(s) (for text annotated from the web page) as well as ann:ImageAnnotation(s) (for future extensions with image media)

The textual annotations also keep track of the different possible lexical realizations (ann:text property) that a same object may have exposed into different web pages: they are not addressed as alternative labels for the resource, but are uniquely associated to that specific annotation, since they may also refer misspelled entries or other kind of references which the user may not want to associate to the targeted resource. The annotated text is used to retrieve the textual occurrence of the resource when the user gets back to the same page (an highlighter icon in the bottom will show the presence of previous annotations on a page and will allow the user to view them highlighted).

The Application Ontologies layer is not limited to include the sole Annotation Ontology, and can be dynamically extended to host new application ontologies according to the needs of future Semantic Turkey extensions (see extension mechanism in the following section).

7. Architecture

The architecture (Fig. 4) of Semantic Turkey follows a three layered design, with the presentation layer embodying the true Firefox extension and the other two layers built around java technologies for administering the business logic and data access.

7.1. Architectural Layers

The following paragraphs describe more in detail the three layers which constitute the architecture of Semantic Turkey

Presentation Layer. Everything relating user interaction is directly managed by the Firefox extension, thanks to a solution directly integrated in the browser. The presentation layer has been developed through a combined use of the XML User Interface Language XUL [30], XBL [31] and Javascript language, which represent the standard technologies for developing Firefox extensions. An XPCOM [32] component has been developed around the Simile Java-Firefox technology [33] to link the presentation layer to the service layer, which is implemented in java. This direct link is actually performed just to wake up a java web server (the java embedded Web Server “Jetty” [34]), which will accept any further communication from the client. This layer is actually not limited to presentation responsibilities, since much of the web related processing (such as accessing pages, navigating their content, extracting portions of text etc…) can be delegated to the web scripting engine of the web browser (in this case, the SpiderMonkey javascript engine of Firefox: http://www.mozilla.org/js/spidermonkey/)

Service Layer. This layer offers services which may be invoked through XMLHttpRequest(s): the server receives the requests coming from the client by
GET or POST http calls, carries out the operations associated to these calls, and replies with an XML response following the Ajax [35] paradigm, so that, independently from the workload that is imposed to the server, the browser can continue to respond to the user. This is a crucial issue for the usability of the application: expensive computations blocking the browser would otherwise not be tolerated by the user. Besides supporting the communication with the client, the middle layer provides the functionalities for definition, management and treatment of the data and the business logic or applications built on top of Semantic Turkey framework.

**Data layer.** It is mainly constituted by the component for managing the ontology. This has recently been rewritten as a series of dedicated middle-layer API for accessing ontological data: these offer both RDF triple-level access methods as well as more object-oriented tailored facilities, which have been appreciated in RDF libraries like Jena [36] (more details in the following section).
7.2. The extension mechanism

Semantic Turkey features an extension mechanism based on a proper combination of the Mozilla extension framework (which is used to extend the user interface, drive user interaction, add/modify application functionalities and provide javascript API for the whole set of Mozilla desktop utilities) and the OSGi java extension framework [37] which provides extension capabilities for the service and data layers of the architecture.

OSGi Extension Points

OSGi compliance is obtained through the OSGi implementation developed by the Apache Software Foundation, called Felix (http://felix.apache.org/). Three main extension points have been introduced: a Service Extension, an OntologyManager Extension and a Data Extension.

The first one allows for the development of arbitrary services which can be added dynamically to the system. Extensions of this type typically need to realize both a client extension through Mozilla technology, by adding new functionalities (and hooks for them in the user interface) to the system, as well as a service extension which is added dynamically through OSGi.

A extension point extending the previous one allows for the registration of plug-ins: these act as collectors for set of services sharing a common logical ratio. While standard service extensions are sort of add-ons to the main application and are always available unless deactivated or removed, extensions bound to plug-ins are activated/deactivated all together according to the status of the plug-in. Plug-ins are assigned to projects and their status and persistent information is stored with the metadata for each project.

The OntologyManager extension provides openness to different triples store technologies. The extension mechanism is based on the exploitation of a middle-layer API for RDF access: OWL ART API4, and on connection methods for dynamically plugging different triple store technologies. The RDF middle-layer API are defined through a set of interfaces, which can be implemented to adopt different triple stores. This can be of particular interest in specific scenarios where the target user has to connect to a specific ontology repository, or where a service extension is being built by annexing an existing application, and in either case, these are based on a different triple store technology. Current implementations provide support for Sesame2 [27], OWLIM [28] (Sesame implementation allowing for reasoning over OWL [38] data), AllegroGraph5, Jena [36] and Protégé OWL API [6].

Finally, a data extension point allows for the declaration of support and application ontologies which are loaded by the system to drive its behavior and the one of its extensions and connected applications. These ontologies are not treated the same way as imported domain/user ontologies (that is, the ones the user is explicitly importing) and are explicitly registered for their role. Registering an ontology through this extension point has a variety of consequences: these are loaded automatically at system startup even if they are not explicitly imported by the edited domain ontologies; moreover, application ontologies’ content (and content classified under application ontologies’ concepts) is not shown explicitly but only managed and exposed indirectly through applications’ services.

We enabled this classification of ontologies since all the data which is available through ST is available as RDF triples: it was thus mandatory to separate the knowledge which is being managed by the user, from the one which is being used by Semantic Turkey to coordinate its activities. Despite this “conceptual separation” – ontology spaces are managed through the use of named graphs [39] – having a single RDF cauldron where all triples are being stored allows for more tight connection between these spaces, so that, for example, data in the application space could be used to organize (classify) the domain information according to different facets, or to add annotations which should not be available in the domain space. As an example of application ontology, ST declares an application ontology called Annotation6 (see section 6 on Knowledge Model) describing the textual occurrences from which entities submitted by the user have been annotated, together with details about the document (type of document, url for web pages, title etc…) where these annotations have been taken. An example of support ontology is instead provided by the Sesame2 implementation of the OntologyManager extension point: Sesame2 library does not support OWL reasoning nor includes the OWL vocabulary; since ST relies on the OWL vocabulary, this is being declared as a support ontology and dynamically added to the core knowledge any time an OWL Project is being created.

4 http://art.uniroma2.it/owlart/
5 http://www.franz.com/agraph/allegrograph/
6 http://art.uniroma2.it/ontologies/annotation
Data defined upon vocabulary from the Annotation ontology (since it is an application ontology) is thus not shown by default in all ontology editing interfaces, and its content is made available to the user through ST’s functionalities (such as those for retrieving documents associated to ontology resources, or for highlighting all the annotations taken in a document), while resources from the OWL vocabulary (being it a support ontology) are shown but are kept separate from user data (owl vocabulary is not saved together with user data nor need to be explicitly imported by user ontology).

Extensions implementing at least one of Service or OntologyManager ext. points (extending the KM obviously only makes sense if accompanying further functionalities or different persistence models) can be deployed as xpi (cross-platform installers) packages. Once installed inside Firefox, extensions expose the client part (the true Firefox extension) as usual, while their installation directory is managed by Semantic Turkey Extension Discovery System, which extracts OSGi bundles and installs them in the main application. This assures an easy and fast process for the user, which can install ST extensions as any other Firefox one, by dragging the xpi over Firefox and restarting the browser.

Client Extendibility

The client part of Semantic Turkey can be extended through traditional Mozilla overlaying mechanism, which allows for new extensions to add graphical elements (and associated javascript functions) to existing browser sections (such as the sidebar, the main browser, the menu etc...). The overlaying mechanism also allows to override existing elements by replacing their semantics with new ones provided by introduced extensions. However, the extension points approach is preferable to overlaying in those cases where a given functionality is foreseen in the basic system and is thought in advance to be extended by many alternative applications. This is the case, for example, of the annotation mechanism, which is available in the basic system, but could be extended as well, allowing new annotation functions to replace those normally associated with the drag’n’drop gestures. An overlaying of the annotation handlers would not be desirable, since new annotation extensions would override each other and reverting to the original one would require forcing their deactivation. To overcome these limitations, extension points can be set also in the Mozilla Framework by developing XPCOM components\(^7\) or dedicated modules\(^8\) for registering extensible functionalities, and for letting the user choose the one to be active.

JavaScript API for accessing Ontology Services

Upon the above layer, a set of JavaScript API\(^9\), completely hiding the HTTP request/response interaction, has been built by using Mozilla technology. These API are coded as exportable functions into Mozilla modules, and they hide HTTP communication details through hiding/encapsulation (a module’s developer must choose which objects/functions are exported by users of the module and which ones just serve as hidden internal machinery).

These JavaScript Modules (roughly paired with their service counterparts in the service layer) can thus easily be imported into any sheet of any Mozilla based application (not necessarily Firefox, even an email browser such as Thunderbird, if properly interfaced with ST’s service layer, can be adapted to rely on ST’s semantic services). In the following example:

```
Components.utils.import(
    "resource://stservices/SERVICE_Cls.jsm",
    art_semanticturkey
)
```

all the objects and functions exposed by the SERVICE_Cls module are imported into the variable semanticturkey: this is a good practice to prevent variable clashing, as Mozilla extensions share a common space where all script code (from main application and all of its extension) is pooled.

Once the above statement is uttered in a script document, API methods contained in SERVICE_Cls can be used in the same sheet, like in the following:

```
semanticturkey.STRequests.Cls.getInstanceList(clsName)
```

where all instances of class identified by clsName are retrieved and returned by the method.

HTTP masking is handled by a common module:

---

\(^7\) XPCOM components are cross-platform components able to provide persistent objects/services to the platform. They are called cross-platform because they can be implemented through different technologies and then provide data through methods declared on a CORBA-like interface

\(^8\) modules are a proprietary Mozilla solution for JavaScript allowing for persistence: JavaScript objects inside a module persist upon different imports of the same module (their use is simpler with respect to components, and is preferable to components whenever object/service implementation is in pure JavaScript)

\(^9\) http://semanticturkey.uniroma2.it/documentation/jsdoc/
which is shared by all API methods. The SemTurkeyHTTP.jsm module contains convenience methods for composing GET and POST requests, for unmarshalling received XML/JSON over HTTP responses and recomposing them in terms of dedicated JavaScript objects.

Due to the masking of HTTP details by Mozilla JavaScript Semantic API, all of their methods return explicit JavaScript exceptions. These are classified as:

- **errors**: error JavaScript exceptions mask HTTP communication errors as well as exceptions thrown at run time by the invoked service and caught by the HTTP Server. Usually it is not easy for the common user to discover the problem which has been generated, and these kind of exceptions are considered as severe application faults.

- **exceptions**: JavaScript exceptions marked as application exceptions are due to predictable java exceptions which occurred at server level. Usually they contain understandable messages which may be explicitly communicated to the user. Also, specific management of these exceptions depending on their type and the context where these occurred can be performed by the application invoking the method which threw them.

Developers of new applications based on the Mozilla framework can thus invoke the underlying services and handle exceptions depending on the context of invocation, thus following a traditional structured programming approach and producing readable “narrative scripting” code, instead of writing complex code for client-server interaction.

Application Developers willing to add further APIs for interfacing with their software, can extend the service layer through OSGi and then build new modules for the JavaScript API, relying on the common SemTurkeyHTTP.jsm infrastructure.

### 8. Conclusions

In this issue we presented (in its last official release) Semantic Turkey: a browser extension which can be used both as a Conceptual Bookmarking System as well as a Knowledge Management and Acquisition Platform or as a basis for developing complex Semantic Web applications: we discussed the main innovations introduced with respect to its original prototype and showed the potentialities of this framework by presenting its extensions capabilities.

#### 8.1. Discussion and Lessons learned

The experiences that we have recently undergone in the adoption of Semantic Turkey across different application scenarios have been a test bed for evaluating the real possibilities of such an extensible framework. The result is that, though far from perfect, the extension mechanism (together with the open service based approach) is flexible enough to allow for very different uses of the platform. For example, a few extensions, such as UIMA [40] and STIA [41], showed how it is possible to build completely new tools by working on the web browser side, adding new services (sometimes of consistent weight, such as in the case of UIMAST) or building completely new applications (such as for STIA) solely relying on the ontology management framework provided by the underlying platform. Furthermore, both of them showed how it is possible to incrementally improve the platform by building extensions on top of other extensions (both of the two cited above depend and rely on RangeAnnotator10, which is itself an extension, and UIMAST can itself be extended with new projection facilities).

The above experiences also made us better understand the added value given by the underlying ontology development framework, which comprehends high level data access and manipulation primitives going beyond basilar RDF management, as it is commonly provided by triple store libraries/services such as Jena or Sesame.

Finally, one more lesson gained from these experiences is that the learning curve for extension developers is a bit steeper due to the wide range of employed technologies and to their different levels of integration: this will require even stronger attention on solutions and support for an Aided Extension Development, which goes beyond extensive documentation and probably embraces the realization of dedicated tools and development frameworks. Enabling and supporting the growth of a dedicated open software development community has been in fact one of the key aspects in several successful experiences (e.g. Protégé).

---

10. [http://semanticturkey.uniroma2.it/extensions/rangeannotator/](http://semanticturkey.uniroma2.it/extensions/rangeannotator/)
8.2. User Evaluation

We’re currently conducting evaluation of the tool upon an open questionnaire which is available at: http://semantic turkey.uniroma2.it/questionnaire/

Contributions are still few\(^{11}\) to trace a statistically significant analysis (also because the questionnaire provides different questions depending on the user profile, which may vary from domain expert to Semantic Web Application Developer), though we collected most prominent results (homogeneous across different users) which revealed Semantic Turkey’s strong points and flaws:

- **User interface** is considered friendly. All voted from “satisfactory” to “yes, sure!” upon the explicit question about friendliness of UI, and this has been remarked with comments – especially from domain experts – comparing it to other available tools.

- **Easiness of installation** is another strong point, though someone reported problems – in their comments to the answer – whenever Firefox java plugin is not properly setup: this is not something directly related to what can be done at application level, though we acknowledge that the underlying technology (java plugin, firefox xcom etc..) is not completely 100% guaranteed to work immediately on all machines, and may require some setup\(^{12}\)

- **Extension Development**: the very few users who completed this part (rating themselves as Semantic Web Application Developers) rated the Extension Development learning curve as steepy, thus confirming our considerations in previous section, though half of them really appreciated the possibilities of mixing different technologies and saw the learning phase as the necessary cost to pay for getting to them

- **Semantic Bookmarking and Annotation**: the bookmarking feature of ST is seen as an added value wrt existing tools (again, domain experts with no high computer skills provided most of the positive feedback) though some of the users complained about lack of other bookmarking possibilities, such as bookmarking concepts other than instances: outside of the questionnaire, this feature has been requested to us especially by researchers working on Semantic Annotation, in need of providing training datasets of pages tagged with respect to both entities and concepts.

8.3. Future Research Work

Probably, the next step which further research and development on this platform should take is to address the potentialities which have arisen by opening it up to full ontology development. In its new incarnation as a platform for development and acquisition of Semantic Web Data, we cannot ignore important modeling axioms provided by the OWL language (restrictions, set operators etc…which are currently not available for editing, though being properly processed by the data&inference layer), and include explicit support for different modeling frameworks, such as SKOS\(^{13}\).

On the other hand, while the above aspects are important in ontology development systems, there are other directions that, being by far more concerned with the contradistinguishing features of ST, could be properly investigated to push forward state-of-art research on this kind of framework. The presented architecture, thanks both to its modularity and web interaction features, could naturally aspire to a collaborative framework allowing knowledge engineers and domain experts to exchange information, opinions and data over the same working environment.

Identification of proper user roles in the acquisition and development process could then give rise to a whole range of dedicated services which would easily be activated/hidden depending on the profile of the logged user. We are currently pursuing this objective, by introducing concepts which are closely common to (and inherited from) traditional solutions in Software Engineering, such as Bug-Tracking and Discussion, Issues Management, Versioning etc…

Another research line which naturally follows from the intrinsic connection between ontology and documents in ST, is related to the elicitation of knowledge from (web) resources: we are currently studying processes for automatically extracting knowledge from documents (and any kind of unstructured information available from the web), proactively collabor-

\(^{11}\) We left the questionnaire open to the community and avoided any artificially induced experimental setting (such as evaluation by our students and the like)

\(^{12}\) Though these rare issues mostly affect Linux machines with non-SUN JVMs or not properly configured JVMs, thus happening to people with averagely more-than-average computer skills who know how to setup their system

\(^{13}\) Both these features are currently being introduced in the platform. In particular, SKOS/SKOS-XL editing will be available in the next version to be released in september 2010
rating with the user on how to use the collected information for populating/enriching managed ontologies. A first approach has been presented in [40]. Finally, we found many overlapping points with current research on Semantic Desktops, especially in those modeling aspects which have been widely discussed and synthesized in the PIMO Ontology for Personal Information Models [42]. Interaction with this research field could be two-ways: by exploring assessed results in Semantic Desktop research, to better handle knowledge organization inside the current platform, as well as by transforming ST into an end-point for Semantic Desktop interaction. Semantic Turkey site can be reached at: http://semanticturkey.uniroma2.it/

References

[27] Jean Broekstra, Arjohn Kampman, and Frank van


[38] W3C. [Online]. http://www.w3.org/TR/owl-features/


Notes for reviewers

Testing the tool for the review

Taking into consideration comments posted on page: [http://www.semantic-web-journal.net/content/special-call-semantic-web-tools-and-systems](http://www.semantic-web-journal.net/content/special-call-semantic-web-tools-and-systems) of the SWJ for this special call, reviewers are encouraged to review both the paper and the tool by downloading it from its home site and testing it. Though a result of research effort, the tool is stable and mature (and used by hundreds of users). We only report two small bug issues in the current version which have already been solved but which, for better organization of releases, will be included in the next release (which will feature also other major improvements such as SKOS editing and better event management):

1) SPARQL querying: language tags are not properly managed for literals in the result page of a SELECT query (this may prevent the table to be shown at all)

2) UNICODE: we discovered that the java virtual machine started by the java plugin framework inside web browsers lacks lot of the configuration values provided wrt the one ran in the standard way. This may cause problems when handling UNICODE strings (dragged and dropped from the text to the ontology) unless (by accident) the system default encoding is not able to natively support those character glyphs. We solved this problem in new (not still released) version, by changing handling of UNICODE strings and making it independent from JVM configuration parameters

3) Semantic Navigation has been temporarily disabled in current version due to deep changes in the triple API

Also, the tool is very easy to install (just a drag’n’drop of the xpi over firefox), though, misconfiguration of Java-Plugin on Firefox (expecially on Linux systems, which may come with different installed JVMs other than the one distributed by SUN, such as OpenJDK/Iced-Tea) may cause problems which are not related to any malfunction in ST. In the rare case that something went wrong, you may check the page: [http://semanticturkey.uniroma2.it/documentation/installation.jsf](http://semanticturkey.uniroma2.it/documentation/installation.jsf) and also this (also linked by previous one):

http://semanticturkey.uniroma2.it/documentation/ffox-java_help.jsf

The authors are also available for helping in solving any issue related to that.

Paper length

We exceeded the paper length proposed in the special call (8-10 pages). We asked what to do to the editor who suggested (obviously with no other responsibility on the outcome of the review) not to cut the content too much and just trade between synthesis and exigencies in describing our work. So we gave it a first cut (removed description of most notable extensions) and just left a few pages exceeding the maximum reported length of 10 pages (currently 15). Possible further cuts will be discussed in case the paper is being accepted, and any suggestion provided by reviewers is welcome.